

<p>MISSION EN ENTREPRISE</p>
------------------------------

**MASTERE : INTERNET ET SYSTEMES REPARTIS**

<p><b>TITRE DE LA THESE :</b></p>
-----------------------------------

<p><b>AN AUTOMATED BENCHMARKING TOOLSET : EXPERIMENT DESIGN AND CONTROL, RUN TIME SUBMODULE.</b></p>
--

**Nom de l'entreprise : National Institute of Standards and Technology**

**Ville : GAITHERSBURG**

**Pays : ETATS UNIS**

**Dates : Période de stage du lundi 3 mai au 31 décembre 1999**

**Réalisé par : M. Benjamin TRAVERSE**

**Sous la direction de : M. Alan MINK**

**(Directeur de stage)**

**Et de : Mme Monique BECKER**

**(Conseillé d'études)**

---

---

Prénom et nom de l'élève  
*Benjamin Traverse*

Mastère *ISR*

.....  
.....  
Nom de l'entreprise d'accueil  
*National Institute of Standards and Technology (NIST)*

.....  
Lieu du Stage  
*GAITHERSBURG (Maryland, USA)*

.....  
Nom du conseiller d'étude (enseignant INT)  
*Monique Becker*

.....  
**TITRE DU RAPPORT**

.....  
**AN AUTOMATED BENCHMARKING TOOLSET:  
EXPERIMENT DESIGN AND CONTROL, RUN TIME SUBMODULE**  
.....

**RESUME:**

L'évaluation des performances des ordinateurs est une tâche continuelle et coûteuse en temps et qui génère de grandes quantités de données à stocker et analyser. Le but de ce travail est de proposer une méthode automatisée pour générer, mesurer et analyser un profil exhaustif des performances. Bien que nos efforts soient portés principalement sur des réseaux de machines, cette méthode s'applique à tout système informatique parallèle ou distribué. Notre méthode se divise en trois modules distincts : (1) un module de collecte et de stockage des données, (2) un module d'analyse des données et (3) un module automatisé d'exécution et de gestion d'expérience. Ce rapport décrit la partie définition et contrôle d'expérience, et sous-module d'exécution.

**ABSTRACT:**

Benchmarking and performance evaluation of high performance computing are a continuously on-going process that consumes significant stage time and generates large amounts of data to be stored and analyzed. The goal of this work is to propose an automate method to generate, capture and analyze an extensive performance profile. Although our initial efforts focus on commodity clusters, they are applicable to any parallel or distributed high performance computing system. Such an automated system has three main components: (1) a data collection and storage module, (2) a data analysis module, and (3) an Experiment specification and execution modules. The focus of this report is on the third module: Experiment specification and execution.

---

---

**Date**

**An Automated Benchmarking Toolset:  
Experiment Design and Control, Run Time Submodule**

B. Traverse, A. Mink and M. Courson

Information Technology Laboratory  
National Institute of Standards and Technology (NIST)  
Gaithersburg, MD 20899  
[amink@nist.gov](mailto:amink@nist.gov)

**Abstract**

Benchmarking and performance evaluation of high performance computing are a continuously on-going process that consumes significant staff time and generates large amounts of data to be stored and analyzed. The goal of this work is to propose an automate method to generate, capture and analyze an extensive performance profile. Although our initial efforts focus on commodity clusters, they are applicable to any parallel or distributed high performance computing system. Such an automated system has three main components: (1) a data collection and storage module, (2) a data analysis module, and (3) an Experiment Specification and execution modules. The focus of this report is on the third module: Experiment specification and execution.

---

## TABLE OF CONTENTS

### *TABLE OF FIGURES*

<b>1. Introduction .....</b>	<b>1</b>
<b>2. Motivation and related work .....</b>	<b>3</b>
<b>3. Experiment Design Submodule.....</b>	<b>4</b>
3.1. Experiment Specification Package.....	5
3.2. Experiment definition example .....	6
3.3. Experiment Plan Design Package .....	15
<b>4. Run Submodule .....</b>	<b>19</b>
<b>5. Choice and installation of a batch queuing system.....</b>	<b>21</b>
5.1. Introduction .....	21
5.2. Choice of DQS.....	21
5.3. Installation, configuration and maintenance .....	22
a. Installation and configuration .....	22
b. Maintenance .....	23
5.4. NT Support for DQS.....	24
a. NTshell .....	25
b. NTbridge.....	25
<b>Conclusions and work in progress.....</b>	<b>26</b>
<b>Acknowledgement.....</b>	<b>27</b>
<b>REFERENCES .....</b>	<b>28</b>
<b>Annex A. Nist Overview .....</b>	<b>29</b>
<b>Annex B. Example of Documentation.....</b>	<b>32</b>
<b>Annex C. NTshell.....</b>	<b>34</b>
<b>Annex D. NTbridge .....</b>	<b>35</b>

---

---

## TABLE OF FIGURES

Figure 1: Project diagram.	1
Figure 2: Process of the project.	2
Figure 3: Experiment Design and Run Submodules.	4
Figure 4: General information.	7
Figure 5: Main window.	8
Figure 6: Application selection.	8
Figure 7: Database applications.	9
Figure 8: Parameters window.	9
Figure 9: List of processors parameter.	10
Figure 10: Communication Environment values.	10
Figure 11: Add a value to a parameter.	11
Figure 12: Choice of values from the Database.	11
Figure 13: Communication Environment values.	11
Figure 14: Value selection.	12
Figure 15: New variable window.	12
Figure 16: New variable “Mynewvar added”.	13
Figure 17: first application stored (“ep” version 2.3b2).	14
Figure 18: After first set duplication.	14
Figure 19: Name of second application has been changed.	15
Figure 20: Experiment Plan Design Package.	16
Figure 21: If a run already exists.	17
Figure 22: Information about the run.	17
Figure 23: Choice of a run in the list of similar.	18
Figure 24: Run Package Status.	19
Figure 25: A pictorial cluster system ancestry from [11]	22
Figure 26: Screenshot from qstat32.	24
Figure 27: Basic diagram of the NT support for DQS.	25
Figure 28: Organization chart.	30

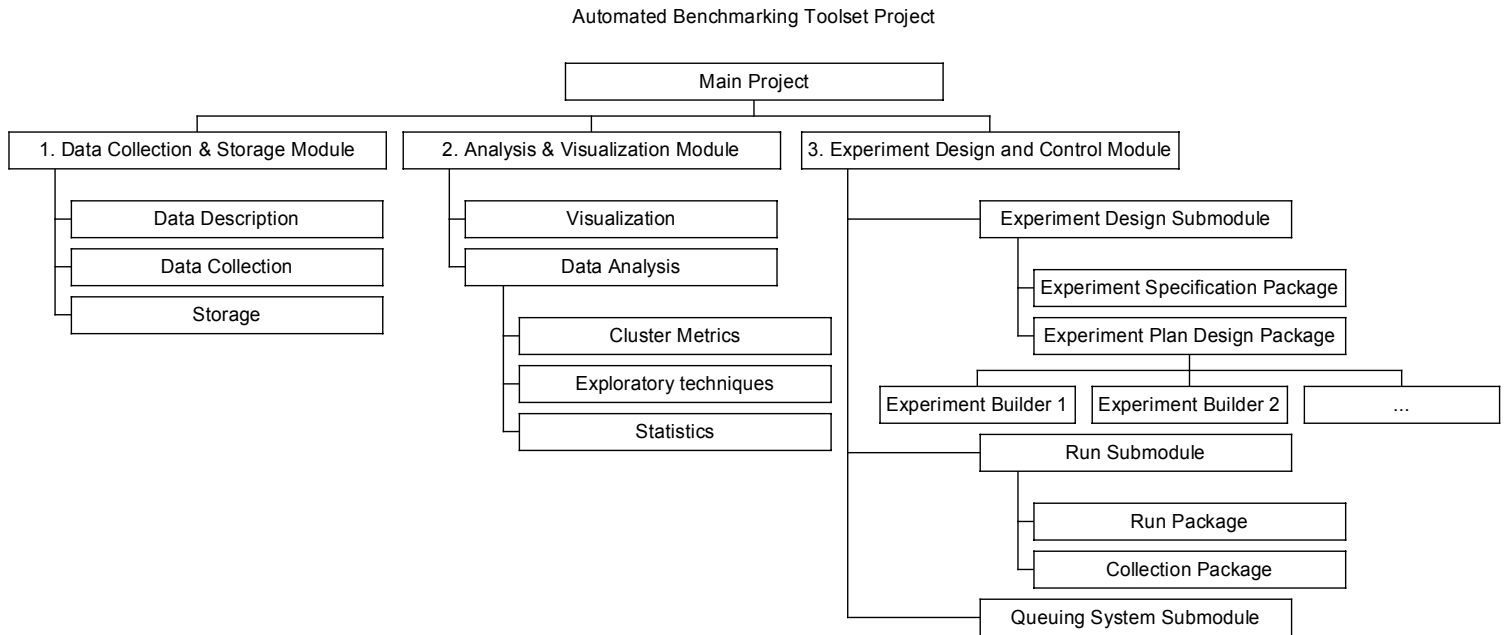
---

# 1. Introduction

The high rate of evolution in computing technologies and the need to have up to date and accurate performance information on parallel computing platforms and applications involve frequent execution of benchmark and application codes. In addition, parallel architecture tuning requires many tiresome manual manipulations that must be repeated often.

The multiple parallel machines and architectures available at NIST (IBM SP2s, SGI Origin 2000s, linux and NT clusters) and especially the three PC clusters in our laboratory (two linux and one NT) provide an interesting base to develop a tool that simplifies, automates and manages the steps required to configure and run these codes as well as collect and store performance information for future analysis.

NIST is developing a prototype of an Automated Benchmarking Toolset which consists of three modules as shown in Figure 1. The first module handles data collection and storage [5]. The second module focuses on data analysis and visualization [6]. The last module deals with Experiment Design and Control and is the focus of this paper. We can further divide this Experiment Design and Control module into three submodules: the Experiment Design Submodule, the Run Submodule and the Queuing System Submodule.



**Figure 1: Project diagram.**

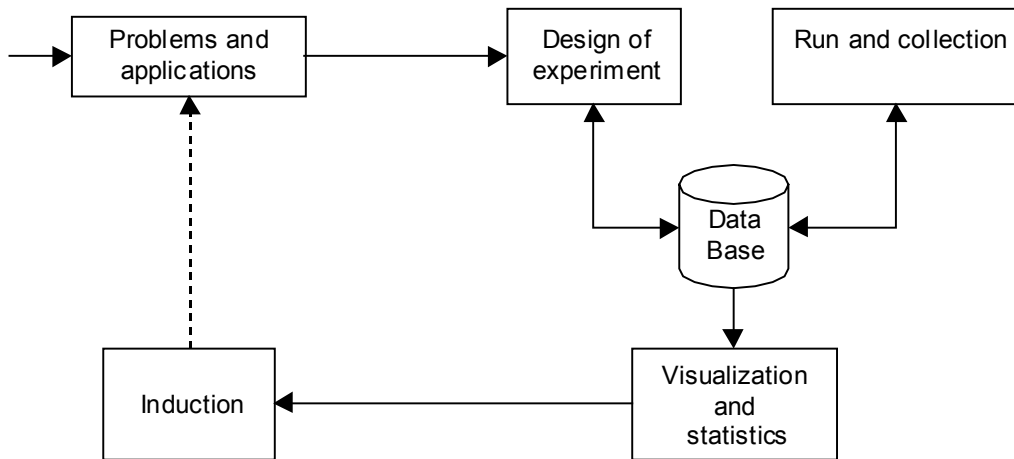
The experiment Design Submodule provides a means to design experiments. The user specifies parameters chosen for study, and then the tool builds an experiment plan, relying on advanced statistical techniques as well as existing data in the database. Then the Run Submodule automatically carries out the experiment plan by submitting only the essential instances to the Queuing system for execution.

Our view of an experiment life cycle is shown in the data flow diagram of Figure 2. The information provided by the user is transformed through several stages of a loop. In the “problems and applications” stage, the user has applications and questions about their performance. So the user defines an experiment based on parameters believed to be sensitive to the performance of these applications in the Design of Experiment stage. An experiment plan, consisting of the description of several application instantiations (or runs) is built and the new experiment plan is stored in the database.

In the “Run and Collection” stage, the instantiations of the experiment plan previously stored are submitted to the queuing system. After the execution of the experiment is completed, the performance data is collected and entered into the corresponding tables of the database.

The “visualization and statistics” stage allows comparison of the execution results in the database to answer the initial questions about the applications in the form of computed metric, plots and graphs. It is also possible to combine data from several experiments for extended analysis.

The last stage of the loop (Induction) is where the user develops conclusions about the experiments, can make various changes to the application or system and then repeat or devise new experiments.



**Figure 2: Process of the project.**

## 2. Motivation and related work

Many tools have been developed that focus on performance measurement. Some of the more well known are VAMPIR [VAM99], from the German company Pallas GmbH, for MPI programs, AIMS [YAN96] from NASA Ames Research Center, Paradyn [MIL95] that features performance tracing down to the statement level through dynamic instrumentation and Pablo [AYD96] which is known for its visualization and self-defining format, among other things.

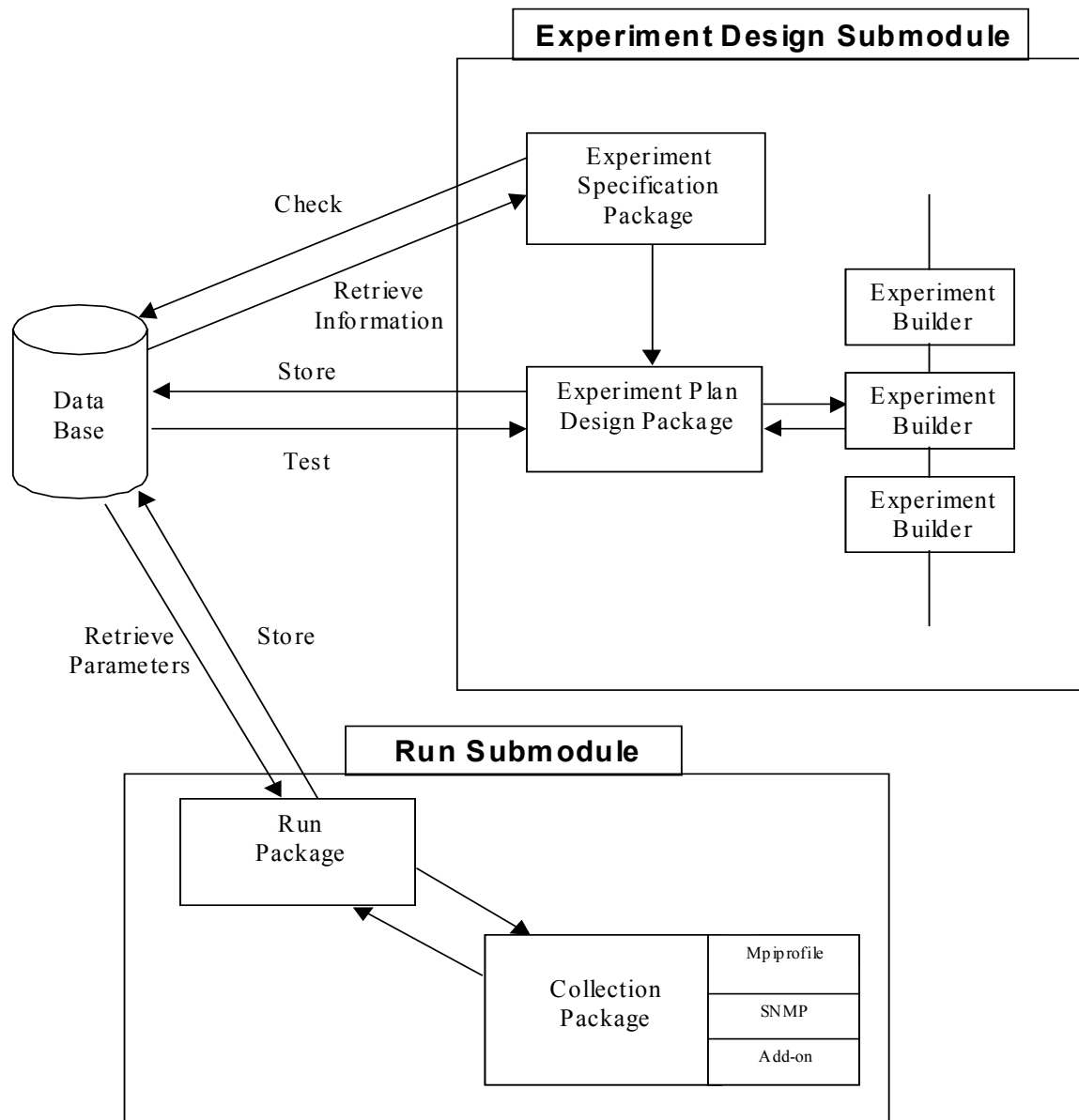
These tools focus much more extensively on instrumenting codes to collect and compute performance measurement metrics and locating computational bottlenecks than our toolset. These and other tools also provide various visualization capabilities.

Although our toolset offers performance measurement and visualization, it does so by incorporating other existing tools, possibly some of the tools mentioned above. In addition our toolset focuses on an integrated database that spans all the “benchmark” codes over an extended time frame. This differs from current effort of the Paradyn team [MIL99] which tracks the performance of a single code with the intention of better tuning of that code. Our toolset will also provide the capability to design and manage the evolution of experiments consisting of multiple codes as well as their input and output files.



### 3. Experiment Design Submodule

As shown in Figure 3, the experiment Design Submodule creates the experiment definition and the experiment plan by providing several packages that guide the user through the experiment construction. The database is a central repository for every package of the submodule, so that the user always has the possibility to reuse data already stored in the database (this is one of the most distinguishing feature of our toolset).



**Figure 3: Experiment Design and Run Submodules.**

The reuse of stored information eliminates the need to repeat experiments, and provides the means to make multi experiment comparison.

### 3.1. Experiment Specification Package

The Experiment Specification Package is the first step of creating the experiment definition. It provides a Graphical User Interface (GUI) which enables a user to build a parameterized shell script, called a Command File, which completely describes the desired experiment sequence for a single application. The function of this specification package is to develop a command file and its parameter values. This information will be passed to the Experiment Plan Design Package where all the combinations of parameter values are evaluated for inclusion in the experiment. The Run Package integrates this information, submitting each combination of parameter values into the command file to produce separate instantiations of shell scripts that will be submitted to the queuing system for execution. Without this tool a user would normally have to write a script manually for every combination of parameters. Although developing shell scripts from scratch may not be significantly faster with our toolset, but building on previous information in the database to build new scripts or modifying old scripts is more productive.

An example of a Command File is shown on the next page. Parameters in the file follow a syntax of a name delineated by percent symbols such as %parameter name%. Examples of parameters in Figure 8 are %nprocs%, %appname%, %iter%, etc. Our interface model is based on developing a set of parameter names, which can be extended, and then assigning a set of values to each parameter name pertinent to this experiment. Parameters with no values are kept in the list, but ignored for the current experiment. To maintain this parameter model we have included the command file as a special case parameter, even though, strictly speaking it is not. We can distinguish two parameter classes: the required and the optional parameters. Initially our set of required parameters are the Command File, the number of processors and of course the application name. Optional parameters can help describe the application configuration (e.g. compiler, link flags and compile flags), the run environment (e.g. Ethernet or ATM Network) or application dependant arguments.

Although the parameter values are text strings inserted into shell scripts, we type each parameter to make it easier for the user to specify the desired set of values. The current types can be a list (zero or more) of integers, strings or floats. Required parameters must have default values. There is a default command file and the default number of processors is one. The application name is currently an exception, where it is the first parameter that must be assigned a value. We are considering revising this to better fit our parameter model and assign a default application. We are currently modifying the package to allow the Command File to be revised, as a special parameter, in this package. Optional parameters don't have default values, if no value is provided, they won't contribute to the experiment.

The following represents our current default parameter list. The first set is our required parameters and the second set is our optional parameters.

Required parameters:

- CommandFile: Specifies the name of the a file that will be used as a command file for the current application execution.
- nprocs: A set of integer numbers, each one specifies the number of processors the application should be distributed among for execution.
- AppName: This is the name of the application to be executed.

Optional parameters (no default values):

- Compiler: The name of the compiler for this application.
- OptimizationLevel, LinkFlags and CompileFlags: These three parameters are compilation options.
- CommEnv: The Communication Environment parameter specifies communication library used by this parallel application.
- Network: The names of the type of communication network to be used by the application.
- Parameters: The information of that field can be used to provide optional parameters required by the command line of the application (typically the size or the class of the problem).

The following Command File example shows how the parameters can be used inside it:

```
#!/bin/csh
# Next line is a DQS directive that provides the name of the output file (processing report)
#$ -N res_%appname%.%class%.%nprocs%.%iter%
# The two next lines are used to place the output files in separate directories.
mkdir /shared/users/lamd/DQS-3.2.7/temp/res_%appname%.%class%.%nprocs%.%iter%
cd /shared/users/lamd/DQS-3.2.7/temp/res_%appname%.%class%.%nprocs%.%iter%
# The next line is a DQS directive that joins the error and output streams in the same file
#$ -j y
# the next line is a comment, in which the number of processors tag will be replaced too.
# want %nprocs% machines
# Next line is a DQS directive that express the hardware requirements: number of processors and
#cluster
#$ -l qty.eq.%nprocs%.and.linux
# next line sets a path for the execution.
set path=( $path /usr/local/lam/bin )
# Next line starts the parallel machine.
/usr/local/lam/bin/lamboot -v $HOSTS_FILE
# Next line is the command line that run the parallel application.
/usr/local/lam/bin/mpirun -v N /home/lamd/%appname%.%class%.%nprocs%
```

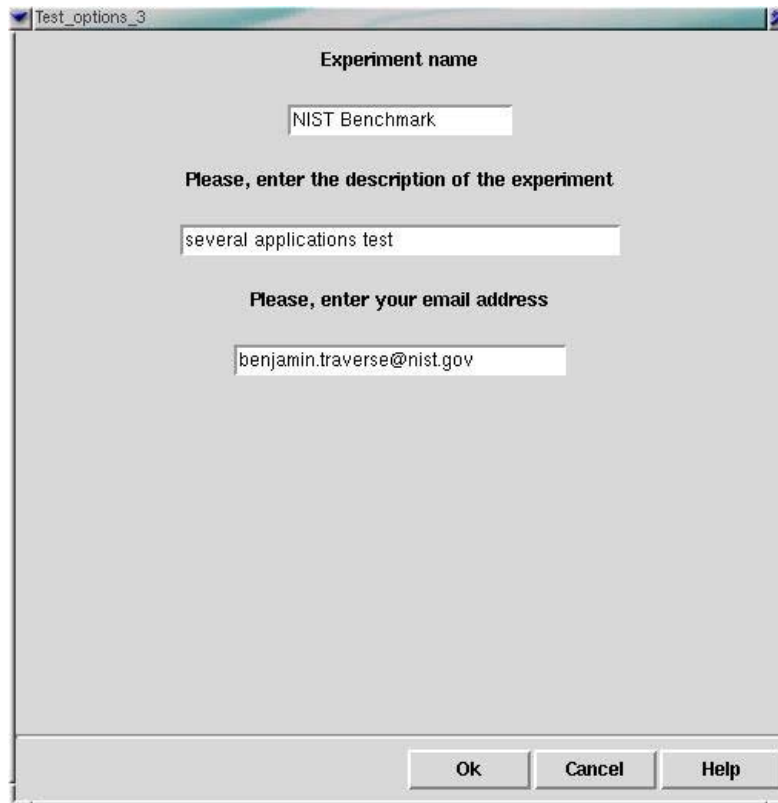
At the end of the experiment specification, a file is produced and sent to the Experiment Plan Design Package. The format of the file is :

```
# date
$ experiment
Experiment name
Experiment description
User email address
$ number of values      parameter name
value1
value2
value3
etc
$ number of values      parameter name
value1
value2
etc
```

### **3.2. Experiment definition example**

To help illustrate the operation of the Experiment Specification Package, the following screenshots show the steps necessary to configure two applications “ep” (version 2.3b2) and “is” (version 2.4b5). Each application requires three parameters: “Communication Environment” with “MPICH” value, number of processors (parameter nprocs) with values from 1 to14 and 16, and an arbitrary parameter “Mynewvar”. The first is an optional parameter retrieved from previous runs stored in the database, the second is a default parameter that is always required and the third is a newly specified parameter.

Figure 4 is the first display after the start of the toolset. All the fields in this form are required to be filled in. As every run of the experiment will be able to be reused in the future, it is important to provide a description to explain the goal of the experiment and a contact for questions about it.



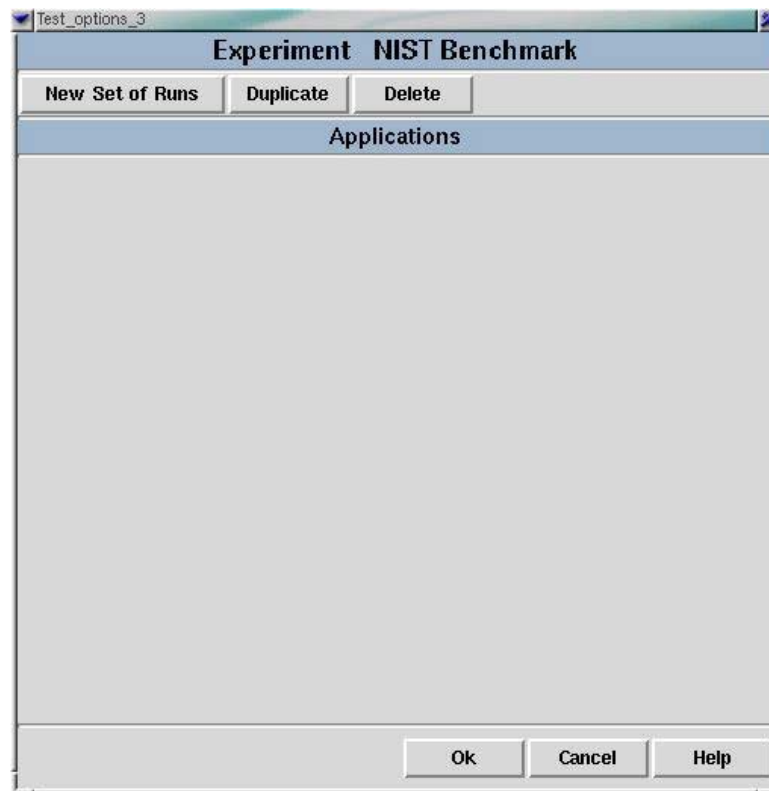
The screenshot shows a Windows-style dialog box titled "Test\_options\_3". It contains three text input fields with the following labels and content:

- Experiment name**: "NIST Benchmark"
- Please, enter the description of the experiment**: "several applications test"
- Please, enter your email address**: "benjamin.traverse@nist.gov"

At the bottom right, there are three buttons: "Ok", "Cancel", and "Help".

**Figure 4: General information.**

Once the information is entered, selecting “Ok” generates the next window which lists the applications of the experiment; initially empty, as in Figure 5. The first step is the selection of a new set of runs (linked to an application) by pushing the “new set of runs” button.

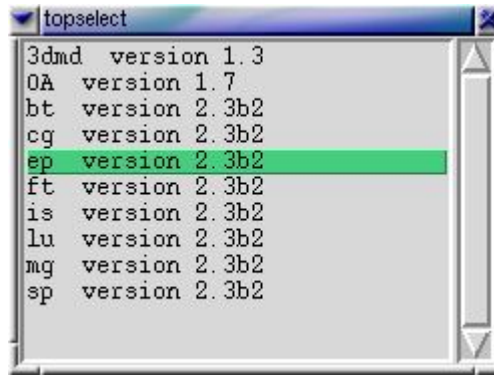


**Figure 5: Main window.**

Figure 6 appears and asks for the application name and version. The user can enter one application name and version manually or select “Browse Database”. To generate a window listing all known applications and versions existing in the database, like Figure 7 (the darkened line is the cursor).



**Figure 6: Application selection.**



**Figure 7: Database applications.**

Once the user has selected an existing application by clicking on its name, or entered a new application name, the user selects the “Ok” button and the next window pops up. After an application is selected, a new window is generated listing all the known parameters from the database for this application, as shown in Figure 8. If the application doesn’t exist in the database, a list of default parameters is displayed.



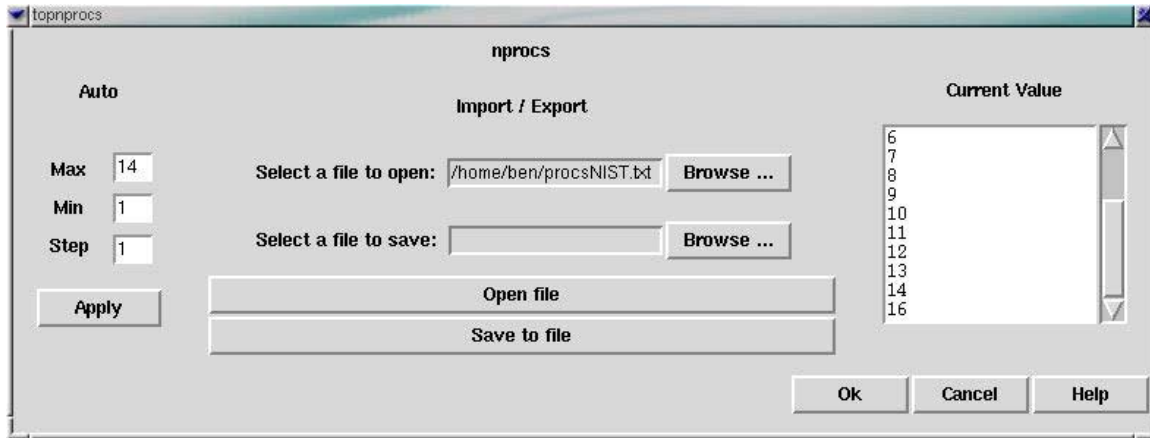
**Figure 8: Parameters window.**

When one of the parameter buttons is selected, another window, which depends on the nature of the parameter, appears. For example, Figure 9 shows the nprocs (number of processors ) parameter window. There are three ways of defining the list of values in this case:

- Fill the Max, Min and Step fields and push the “Apply” button (for geometrical generation of values). The corresponding values are written in the text box on the right side of the window which always displays the current parameter values.
- Load a file containing the desired values: select “browse” and choose a file on the hard drive, then select “Open file”. The values in the file are loaded and written into the text box.

- Enter each value manually in the text box.

Of course the user can combine these steps such as start with a list then modify it manually in the text box to obtain the desired values. To provide the set of processors wanted (1-14, 16), the simplest way is to generate the list from 1 to 14 automatically, and then add 16 by hand at the bottom of the list. Then the user selects “Ok” to record the set of processor values and the display returns to the parameters window.



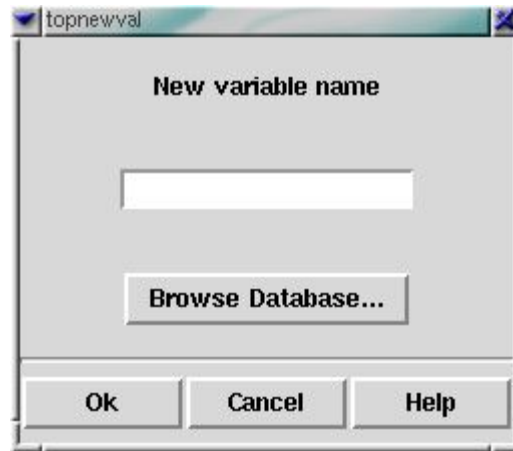
**Figure 9: List of processors parameter.**

The number of processors parameter window is an example of a list of integer values. An example of a list of string type parameters is shown in Figure 10 for the Communication Environment parameter window. The value “MPI LAM” has been automatically provided by the database and indicates that at least one run has already been processed with this value for the Communication Environment parameter.

The user has the capability to select this value by clicking on the gray square on the left of the value name to use it in the experiment, or to add a new value (or to do nothing if he doesn’t want to provide any value for the Communication Environment parameter). The user wants the value “MPICH” for this parameter, so it must be added to the set of available values. To add the new value to the current parameter, push the “add value” button which displays the Figure 11.

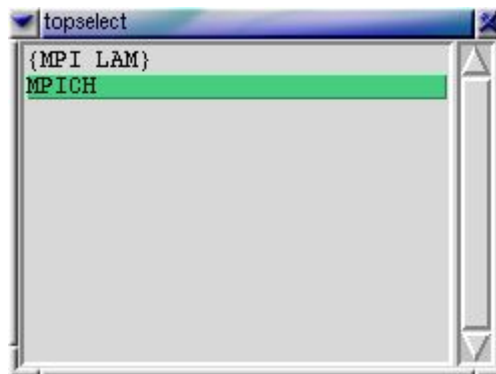


**Figure 10: Communication Environment values.**



**Figure 11: Add a value to a parameter.**

One can enter the new value name manually or click on the “Browse Database” button to display all the values existing in the database for this parameter (not limited to the current application). Figure 12 lists the choices for this parameter in the database.



**Figure 12: Choice of values from the Database.**

The user can select one by clicking on it in the menu (Figure 12) and it is added to the list of values for the current parameter (Figure 13) after a confirmation by pushing the “Ok” button in the Figure 11.



**Figure 13: Communication Environment values.**

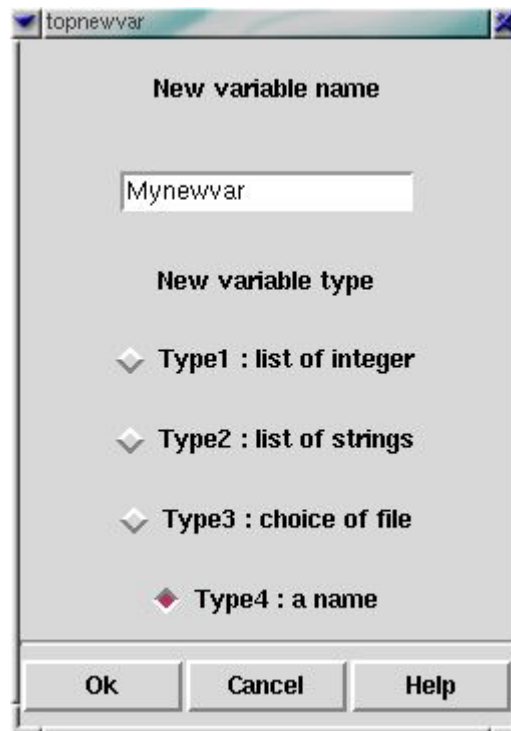




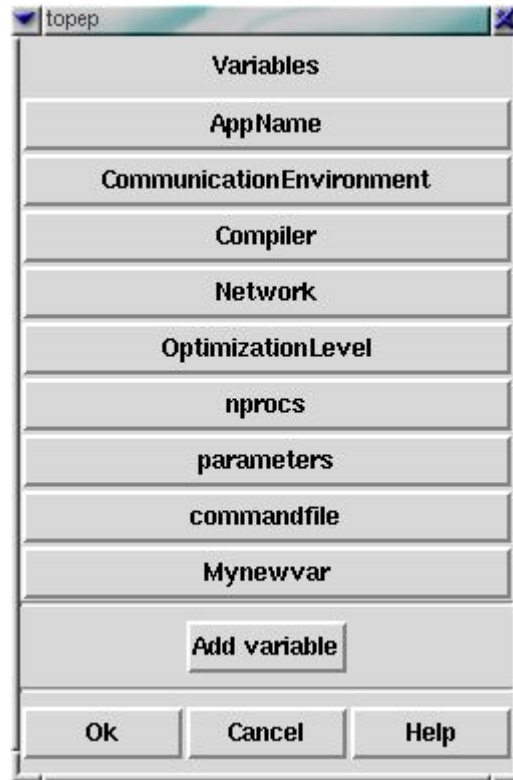
**Figure 14: Value selection.**

The last step to complete the information for the Communication Environment parameter is to select the desired value, in Figure 14, by clicking on the value and then select "Ok". The display returns again to the main parameters window (Figure 8).

A feature of this design is the ability to add new parameters. In the parameters window of Figure 8, the button "Add variables" generates a new window as shown in Figure 15. The process to generate a new parameter is simple: just enter its name and its type. Four types are currently available. When the name and the type of the new parameter are entered, the user clicks "Ok" and a new button for this parameter is created in the parameter window as shown on Figure 16. The new parameter can now be selected to specify new values as described above.



**Figure 15: New variable window.**



**Figure 16: New variable “Mynewvar added”.**

When the user has finished the specification for the current application, he confirms his choice by selecting the “Ok” button in the main parameters window (Figure 16). Then the application name and version are entered into the main window as shown in Figure 17. If the user wants to change anything in this application definition, he simply has to select this application bar to invoke the parameter window.

Many experiments consist of several applications using a lot of common parameters. For example, the NAS Benchmark [8] is composed of 8 applications, each using the same number of processors (2, 4, 8 and 16). It would be very boring to have to redefine every application whereas only the name is different. That is why we’ve added a “Duplicate” button that simply copies a set of runs along with its parameters and store it as a new one, see Figure 18. Then the user only has to change the name of the application by pushing on the new bar button and selecting the application name parameter in the parameters window. If all the parameters are the same between the two applications (except the name), the user simply has to click “ok” in the parameters window and the main window becomes Figure 19. That’s the easiest way to complete the current experiment with the second application.

The last button of the main window is “Delete”. It allows the deletion of any of the application from the list. Of course it doesn’t affect the existing information in the database, as the experiment is not completed yet. The storage part of the experiment only takes place in the next package: the Experiment Plan Design Package.

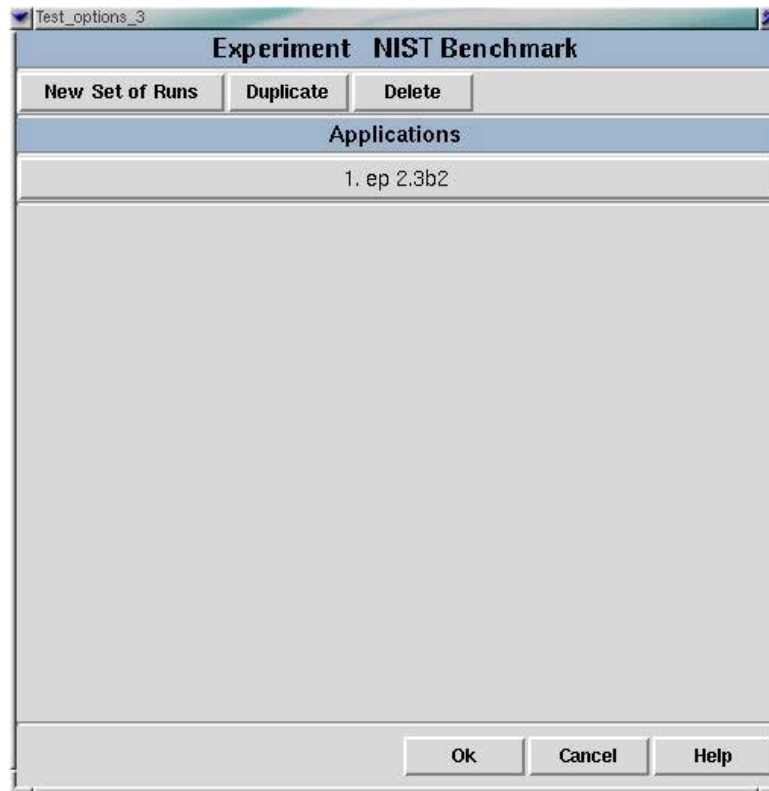


Figure 17: first application stored (“ep” version 2.3b2).

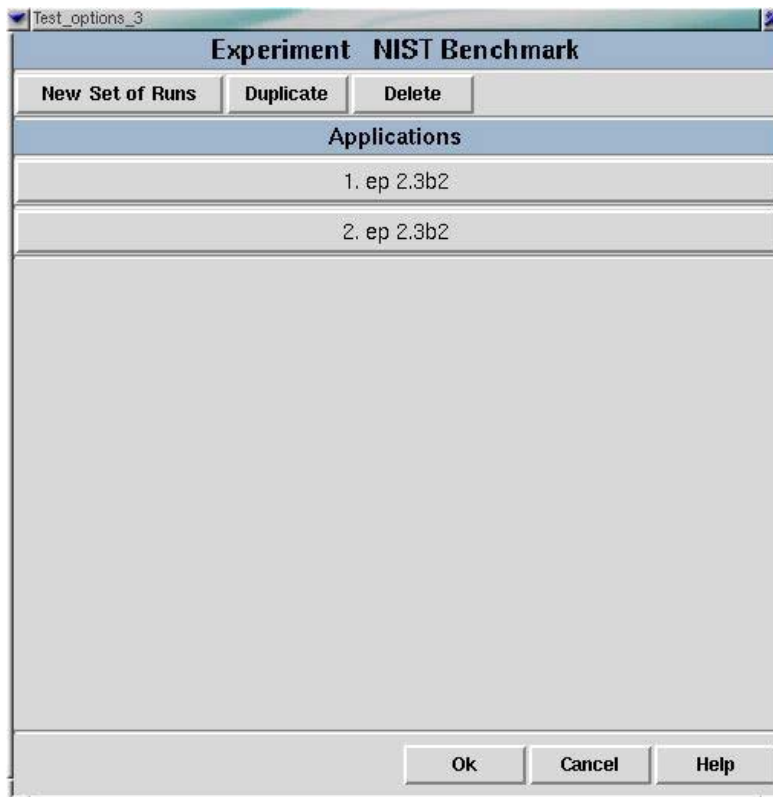
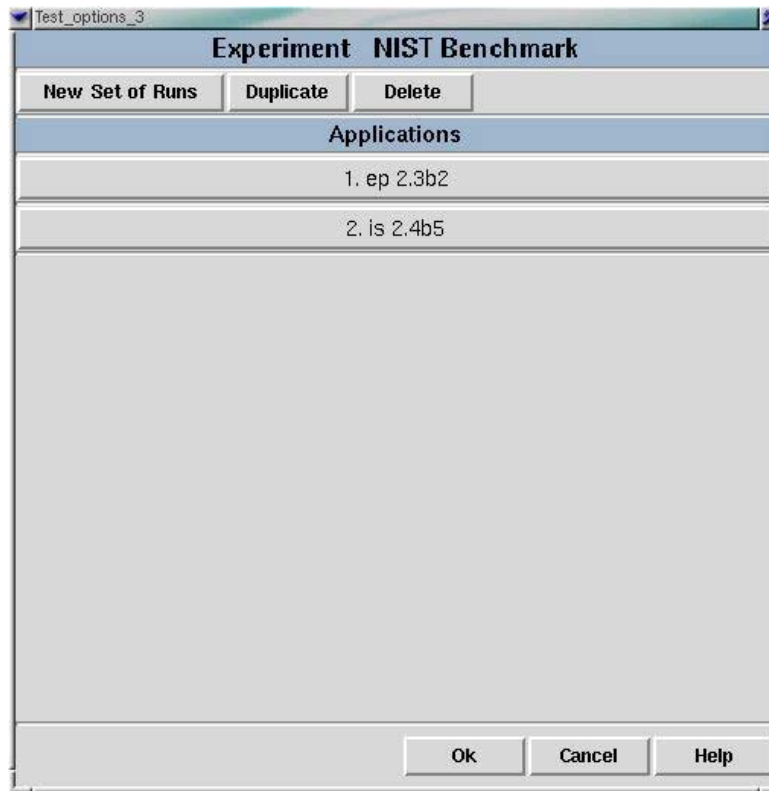


Figure 18: After first set duplication.



**Figure 19: Name of second application has been changed.**

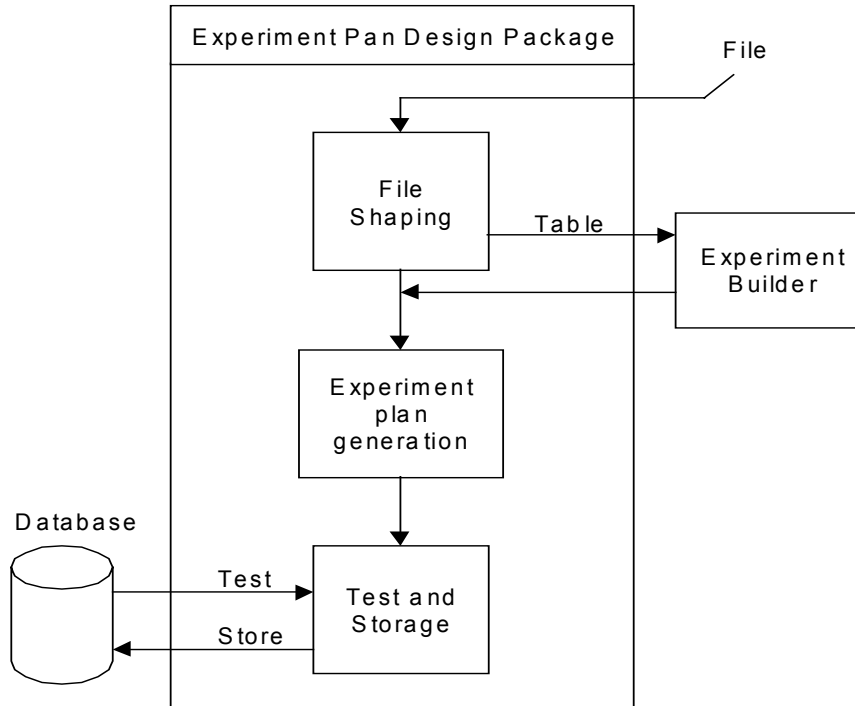
To conclude the experiment definition in this Experiment Specification Package, the user selects “Ok” in the main window and a file is generated. This file gathers the information about the experiment, and is sent to the second package of the Experiment Design Submodule: the Experiment Plan Design Package. The transfer of the file is still manual (the packages are separated and ran sequentially) but the implementation of a program that integrates them will start soon. The first prototype of the Experiment Specification Package is operational and evolving: GUI upgrade, new functionality for better view of parameters values, improvement of the readability.

### **3.3. Experiment Plan Design Package**

The Experiment Plan Design Package builds the experiment plan and stores it in the database. An experiment plan describes the way an experiment should be carried out to provide the requested information. In this case it is a set of fully described runs, that is an instance of an application run. All the parameters specified in the experiment during the experiment specification process are used to produce every combination of values. Designing such an experimental plan follows a well-known process [17], widely used in most scientific disciplines. Based on the requirements (e.g. standard error, interactions), such techniques define the optimal set of runs that match the requirements at the lowest cost (processing time). A number of plan-design plug-ins can drive this package. Our prototype provides full-factorial plans, that select all the possible combinations of parameters (complexity  $O(e^n)$ ), but we expect future support for fractional-factorial plans (up to a  $O(n)$  complexity with no iteration).

Upon completing, we store the fully described experiment in the database as an experiment structure pointing to a list of runs. The runs that are already stored in the database can be reused or re-run as the user desires, whereas new runs are created in the database and flagged for execution by the Run Package. A graphical interface helps the user through the selection of existing runs, providing the opportunity to hand-pick them, let the system chose them or forcibly re-execute the application.

When the Experiment Plan Design Package starts (Figure 20), a file from the Experiment Specification package is passed to it, with the name of the experiment builder chosen. The specified Experiment builder is called and is passed a table corresponding to the current experiment. This table contains the information required to make an experiment plan: number of parameters and number of values of each parameter. The Experiment Builder returns a fully specified experiment plan (a list of fully described runs).



**Figure 20: Experiment Plan Design Package.**

The list of runs is stored in the database. To avoid undesired replication, the package checks if any of these run already exist in the database. If so, it prompts the user for direction, as shown in Figure 21. First, the user has the choice to review the parameters of this run by selecting the “More Info about the run” button which extends the current window as shown in Figure 22.

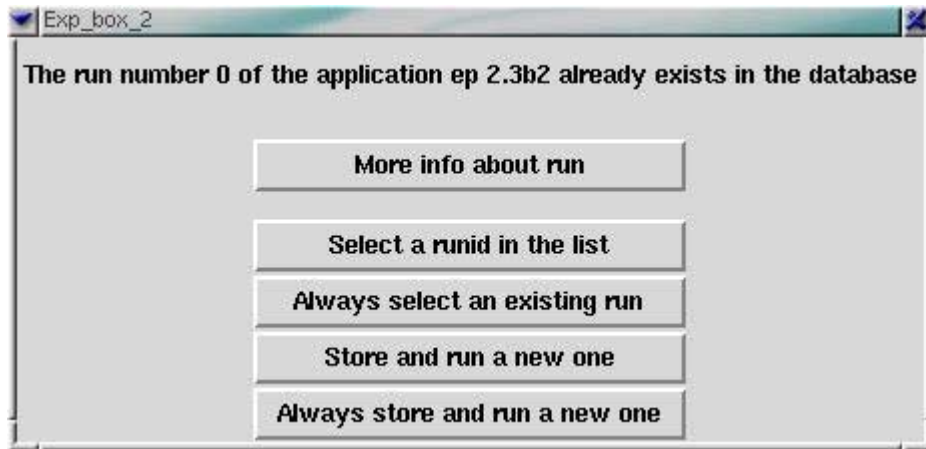


Figure 21: If a run already exists.

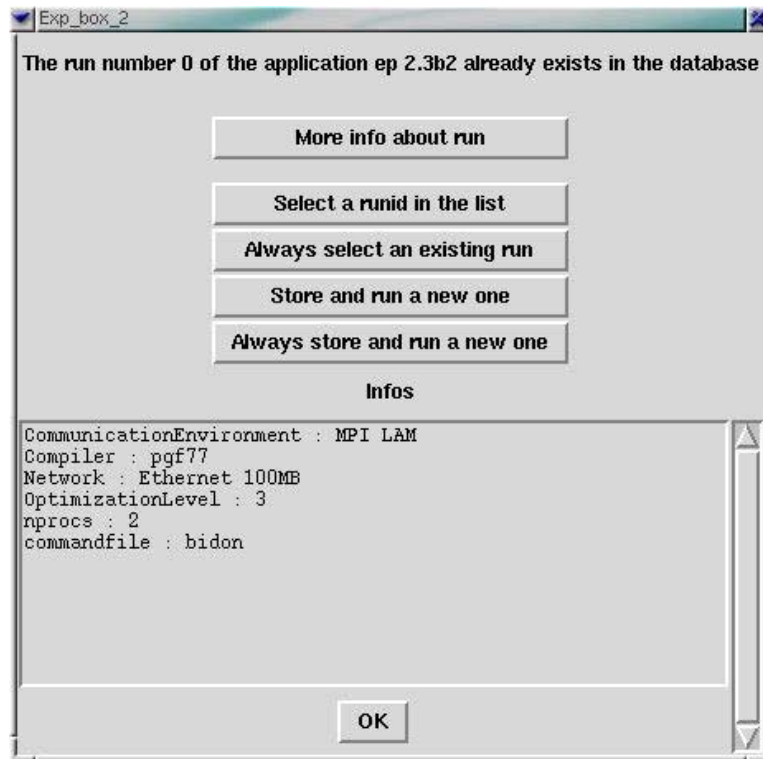
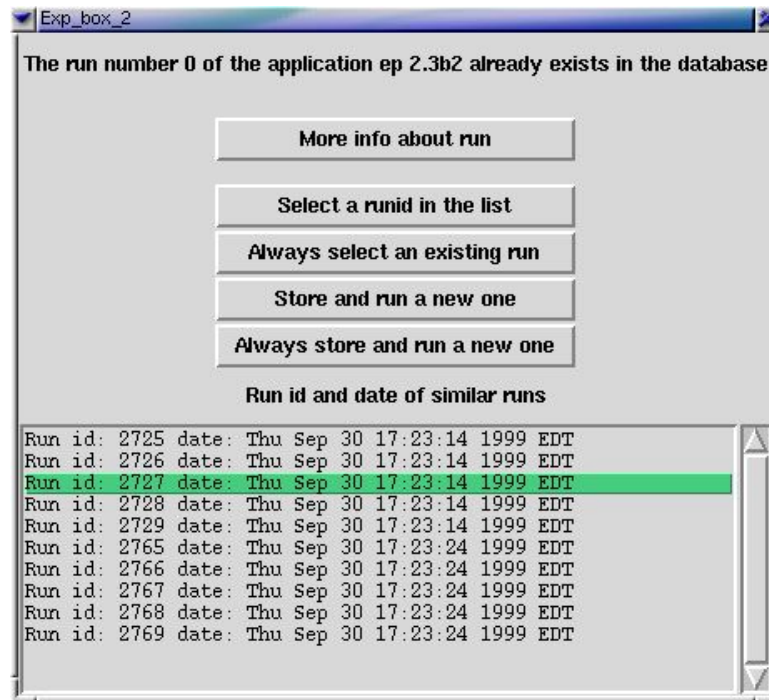


Figure 22: Information about the run.

Four choices are available to the user:

- Select an existing run in the list of similar runs. It means that this run won't be processed another time and only a link to the selected run will be written in the database.
- The second choice (Always select an existing run) lets the system reuse existing results in the database. This is done automatically for every run of the current application (but not for every application of the experiment).



**Figure 23: Choice of a run in the list of similar.**

- The third choice (Store and run a new one) replicates this run, meaning that another occurrence of the run will be processed during the run phase.
- The last button (Always store and run a new one) asks the system to always replicate the runs for the current application.

The process of storing the new experiment is atomic. Thus if a problem occurs during this phase, the storage is not done and the database integrity is maintained. At this point, the application run-time specification physically exists in the database, but is still unprocessed (no performance data). That's the job of the next Submodule: the Run Submodule.

So far, a prototype of the Experiment Plan Design Package has been successfully implemented and modifications are in progress to provide a more intuitive user interface.

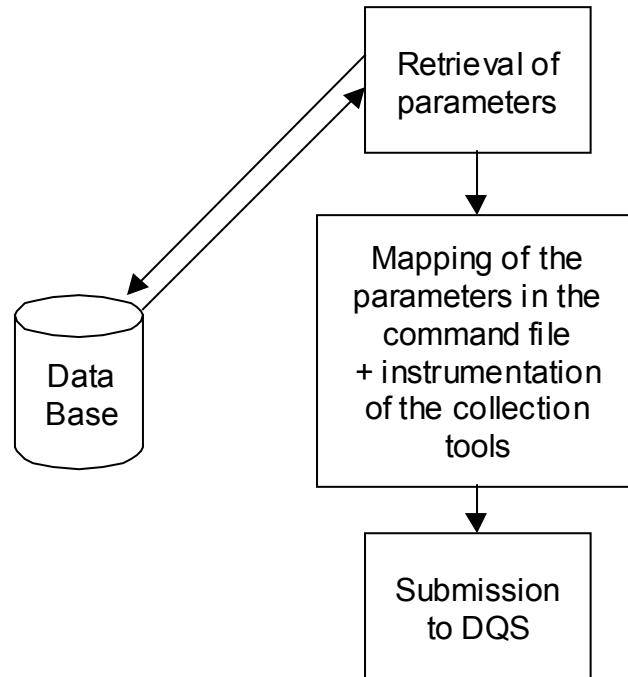
## 4. Run Submodule

The Run Submodule, as shown in Figure 3, consists of two blocks: the Run Package and the Collection Package. The Run Package carries out the individual executions (runs) required in the experiment and manages the performance data collection process (see Figure 24). This component is fully automated. It relies on an existing queuing system for most of the job control. Our prototype uses DQS [7] (see next section). For a given experiment, it accesses the database and retrieves the list of runs to be executed. For each run, it builds a new script file based on the parameters values for this run and the user-supplied command file. The command file is parsed for parameter substitution and instrumented for data collection. The created jobs are then submitted in a single batch to the queuing system for execution. The instrumentation within the script file takes care of starting and stopping the collection system tools by adding a line in the beginning and at the end of the command line as well as updating the records in the database (add performance data) once the job is completed.

The Collection Package is designed to handle most data sources. Our reference implementation currently supports several mechanisms at different levels in the parallel machine.

For low-level data such as memory usage, paging activity or network usage at the process level, we use a modified version of UCD-SNMP [18], a popular, freely available, easy extensible, SNMP agent from the UC Davis. The agent already supports a very wide range of system-level data such as network activity, via standard interfaces. It was extended to give fast access to common statistics such as load average, process information and memory usage, as well as our custom high-resolution tracing hardware (Multikron [19]) and GPS-synchronized timing facility. This information is collected and stored as time series in the database.

Application level data such as response time or number of processors is collected directly from the Run Package.



**Figure 24: Run Package Status.**

Our tool is ready to support any third-party measurement tools. We demonstrated this by collecting high-level performance data from the MPI Profiling library developed at NIST [20]. This library provides transparent timing and call-count information for most MPI functions (send, receive, etc.) as well as synthetic metrics including computation, communication and response time for MPI applications.



The Run Submodule is currently being upgraded to receive add-on collection tools. The process of script files instrumentation and submission to DQS is already completed. The next version will gather in the same package performance data collection and run package.

## 5. Choice and installation of a batch queuing system

### 5.1. *Introduction*

The submission of parallel jobs in a cluster often implies manual efforts, and doesn't insure resource sharing between the machines. The most common and convenient way to avoid these problems is to use a queuing system, which takes care of distributing the applications among the available processors and managing the execution of these jobs. It is possible to help the user to choose the desired set of computers for the experiment execution by customizing the queuing system (adding new submission rules and features). This section explains the process that led to the choice of DQS [7] and the main points of the installation, maintenance and interaction with the project.

Joseph A. Kaplan and Michael L. Nelson [11] have defined a cluster as “a collection of computers on a network that can function as a single computing resource through the use of additional system management software”. To achieve a unified cluster providing distributed computing services, cluster management software such as a queuing system is required. It addresses several issues: static load balancing to optimize the computer resources, distribution of the jobs among the resources, time limits and scheduling rules, etc. All these features have only one aim: optimize the computing power and provide an interface between the user and the machines that make them look like only one resource.

### 5.2. *Choice of DQS*

There are many queuing systems available, each with their own features and specifications. Figure 25 shows the ancestry of a number of queuing systems from [11] in 1994. Since then many new ones have been created. Nevertheless, most of the queuing systems tested in [11] are still available in updated versions, and the basic features are still the same. The most constraining requirement for our choice was the use of a free and open-source queuing system. These parameters restrained the field of investigation very quickly to two products: DQS (Distributed Queuing System [7]) and PBS (Portable Batch System [16]). In their latest version, these two queuing systems provide almost the same features and both are sufficient for our requirements.

Investigating user experience with each of these two queuing systems, we've found many sites explaining their DQS (or CODINE which is the commercial release of DQS) configuration such as [12] [13], and providing information [14] [11] [15] about DQS installation and configuration. Based on this information, we chose DQS version 3.2.7 (4.x is still in beta) to manage our two linux clusters.

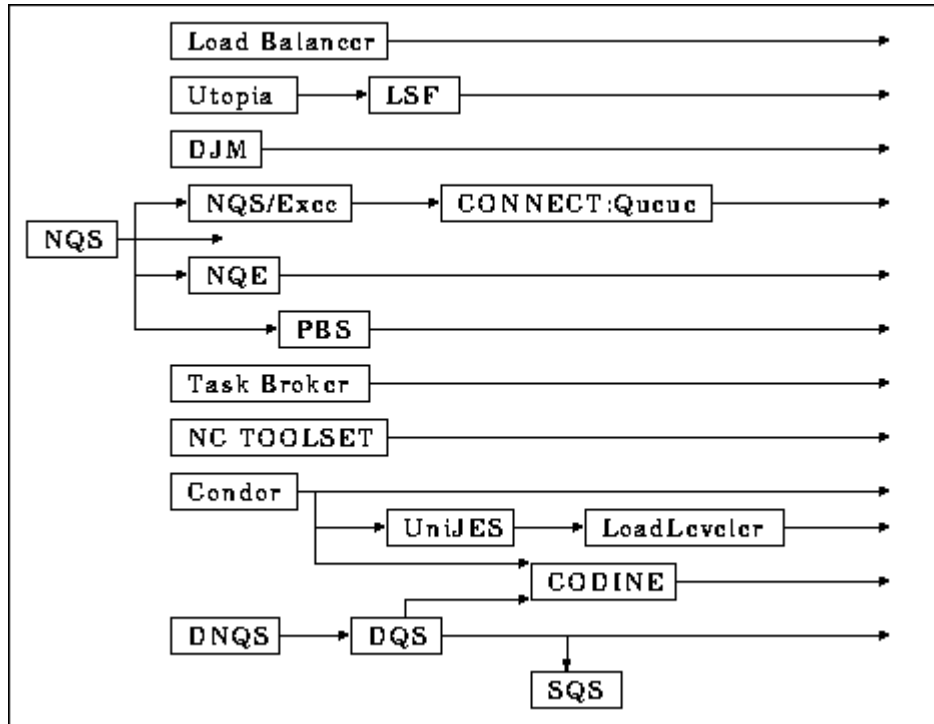


Figure 25: A pictorial cluster system ancestry from [11]

### 5.3. Installation, configuration and maintenance

#### a. Installation and configuration

DQS installation information is available from a number of Internet sites and the DQS support team helps to circumvent missing documentation. A DQS system is composed of a scheduler, the Qmaster, executed in the background on a “master” machine (a safe computer that keeps information about the queues), and by daemons, the Dqs\_execd, executed on every node linked to a queue. The Qmaster manages a list of CPUs, hosts and jobs. When no jobs are running on the cluster, the Qmaster is waiting for messages from the Dqs\_execd daemons. They periodically send information on the computational load of their machine, so the Qmaster always knows the state of resources it is managing. When a user submits a job to the Qmaster, depending on the requested resources and the load on the nodes, it assigns the job to one or several queues. The job consists of a script that specifies its behavior. The script can contain DQS directives (lines beginning with #\$, normalized by POSIX). Then, after the submission with the “qsub” function, the script is analyzed and sent to the Qmaster which adds it to the job list. This list is written on disk to avoid any problem in case of a crash. If the Qmaster has several jobs in the job list, it schedules their execution. The default criterion is the CPU load of the queues (the Qmaster sends the new jobs to the less loaded available queues), but it can be customized easily. The jobs are sorted by priority (if given), and the priorities are changed to avoid any monopolizing of resources by a single user (e.g. after simultaneously submitting a lot of jobs).

The next step for the Qmaster is the allocation of the requested resources (via complexes) if available. A complex is a name assigned to a group of queues to describe the machine resources that host them. It enables the selection of a bunch of machines for execution, based on their hardware specifications. The complexes allocation request is made through a DQS directive in the submission script file. As we are managing two Linux clusters with different generations of Intel CPUs, different memory sizes and one NT cluster, complexes for each of them have been defined to help the user to select the right machines. In the same way, complexes are very useful to distinguish the Linux clusters from the NT cluster, we’ve built a bridge between DQS and NT to enable DQS to manage all of our clusters. Here is the list of the complexes currently defined in our configuration:

To request access to only one of the linux clusters, we defined several complexes based on their processor speed, type, their available memory and we assigned them a name. Thus, to select the slowest cluster, the user may use one of the next complexes: “Linux1”, “speed=200”, “mem=128” or “ppro200”. To ask the experiment to be processed on the fastest cluster, the complexes are: “Linux2”, “speed=400”, “mem=512” or “pII400”. Then to distinguish the linux clusters from the NT cluster, we defined two other complexes: “Linux” that only requests linux machines (in both linux clusters), and “NT” that asks DQS to run the experiment on the NT machines.

The first queue fitting the complexes allocation request is allocated to the job. Until this point, the Qmaster has done all the work. When a queue is allocated to a job, the corresponding Dqs\_execd daemon on the machine handles the execution (if many queues are required for the same job, one Dqs\_execd daemon is chosen to be the master). This machine becomes the master node for this execution and setups the execution environment if the job is parallel (environment variables), then executes the job on the queue(s) allocated (including itself). Before a job execution, DQS can send email to one or several address(es) to tell the user that has started the job execution. DQS has no facility to send email notification that a job has completed execution. Finally, the job is started as a process separated from the master node and the Dqs\_execd daemon goes back into wait state (transmitting the load of the queue to the Qmaster and waiting for other jobs).

The definition of the queues is simple and requires a call to the “qconf” function (see documentation in Annex B for more explanations). The information requested are the name of the queue, the host name and a few other configuration parameters (maximum number of concurrent jobs, time limit for execution, etc). Then to activate the queue, the host must be added to the list of trusted machines, the Dqs\_execd daemon started on the host, and the queue enabled by the administrator. The current configuration of our cluster is one queue per node, with at most one job running on it.

## b. Maintenance

Management of DQS requires only a few functions: qstat, qconf, qdel and qmod. Qconf and qmod are used to change the configuration of the queues. Qstat lists the queues and their state, as shown in Figure 26. Each queue is represented by its name, type, the number of jobs running over the total number allowed (0/1 for an empty queue in our configuration), the computational load of the host sent by the dqs\_execd (“er” means enabled and running), and the state (UP means that the dqs\_execd is working and that the queue is enabled by the administrator).

Queue Name	Queue Type	Quan	Load		State
ntnode1	batch	0/1	0.00	er	UP
ntnode10	batch	0/1	0.00	er	UP
ntnode11	batch	0/1	0.00	er	UP
ntnode12	batch	0/1	0.00	er	UP
ntnode13	batch	0/1	0.00	er	UP
ntnode14	batch	0/1	0.00	er	UP
ntnode15	batch	0/1	0.00	er	UP
ntnode16	batch	0/1	0.00	er	UP
ntnode2	batch	0/1	0.00	er	UP
ntnode3	batch	0/1	0.00	er	UP
ntnode4	batch	0/1	0.00	er	UP
ntnode5	batch	0/1	0.00	er	UP
ntnode6	batch	0/1	0.00	er	UP
ntnode7	batch	0/1	0.00	er	UP
ntnode8	batch	0/1	0.00	er	UP
ntnode9	batch	0/1	0.00	er	UP
pnode1	batch	0/1	0.00	er	UP
pnode10	batch	1/1	0.97	er	UP
lamd SETI	2036 0:12 r	RUNNING	10/25/99 11:20:37		
pnode11	batch	1/1	1.00	er	UP
lamd SETI	2033 0:9 r	RUNNING	10/25/99 11:20:37		

---

pnode12	batch	1/1	0.97	er	UP
lamd SETI	2035 0:11 r	RUNNING	10/25/99 11:20:37		
pnode13	batch	0/1	0.00	er	UP
pnode14	batch	1/1	0.97	er	UP
lamd SETI2	2221 0:18 r	RUNNING	11/18/99 09:48:22		
pnode15	batch	1/1	1.00	er	UP
lamd SETI	1992 0:4 r	RUNNING	10/19/99 00:31:11		
pnode16	batch	1/1	0.98	er	UP
lamd SETI	1998 0:7 r	RUNNING	10/19/99 00:31:15		
pnode2	batch	1/1	0.96	er	UP
lamd SETI	1995 0:6 r	RUNNING	10/19/99 00:31:12		
pnode3	batch	1/1	1.00	er	UP
lamd SETI2	2217 0:16 r	RUNNING	11/18/99 09:48:21		
pnode33	batch	0/1	0.00	er	UP
pnode34	batch	0/1	0.00	er	UP
pnode35	batch	0/1	0.00	er	UP
pnode36	batch	0/1	0.00	er	UP
pnode37	batch	0/1	0.00	er	UP
pnode38	batch	0/1	0.00	er	UP
pnode39	batch	0/1	0.00	er	UP
pnode4	batch	1/1	1.00	er	UP
lamd SETI2	2218 0:17 r	RUNNING	11/18/99 09:48:21		
pnode40	batch	0/1	0.00	er	UP
pnode41	batch	0/1	0.00	er	UP
pnode42	batch	0/1	0.00	er	UP
pnode43	batch	0/1	0.00	er	UP
pnode44	batch	0/1	0.00	er	UP
pnode45	batch	0/1	0.00	er	UP
pnode46	batch	0/1	0.00	er	UP
pnode47	batch	0/1	0.00	er	UP
pnode48	batch	0/1	0.00	er	UP
pnode5	batch	1/1	0.96	er	UP
lamd SETI	2039 0:13 r	RUNNING	10/25/99 15:41: 5		
pnode6	batch	1/1	1.00	er	UP
lamd SETI	1994 0:5 r	RUNNING	10/19/99 00:31:11		
pnode7	batch	1/1	0.98	er	UP
lamd SETI	2034 0:10 r	RUNNING	10/25/99 11:20:37		
pnode8	batch	1/1	1.00	er	UP
lamd SETI	2244 0:19 r	RUNNING	11/29/99 09:41: 9		
pnode9	batch	1/1	1.01	er	UP
lamd SETI	1999 0:8 r	RUNNING	10/19/99 00:31:15		

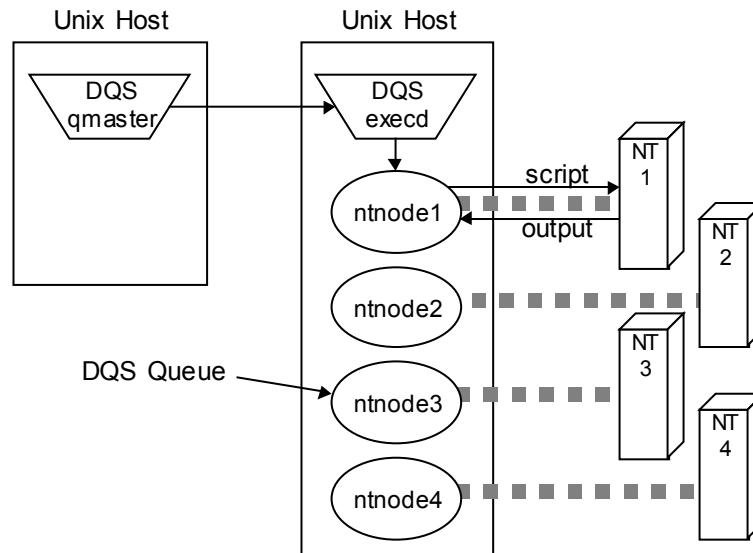
**Figure 26: Screenshot from qstat32.**

In the Figure 26, several machines (in fact one of the two linux clusters) are running jobs. The line displaying a job contains the following information: Name of the owner of the job, the job's name, the job number (sequential count of jobs that have been run from the DQS installation), the state of the job and finally the date and time of the start of execution.

#### 5.4. NT Support for DQS

Although DQS has not been ported to NT (and we do not intend to do so in the near future), we devised a simple mechanism to put our NT cluster under the control of DQS. This mechanism relies on DQS's ability to support multiple logical nodes (or queues in DQS terms) on the same physical host. It means that a single machine can have a `Dqs_execd` daemon managing several queues sharing the machine. Our tool (NTshell + NTbridge) transparently mirrors any job run on one of these virtual nodes onto the target NT host and collects its output. Basically, we are using a linux host with sixteen DQS queues that are

linked to each of the nodes of the NT cluster. When they receives the jobs intended to be run on the NT cluster (from a directive in the command file), our programs (NTshell + Ntbridge) send the jobs to the NT cluster and retrieve the output measurement information and files.



**Figure 27: Basic diagram of the NT support for DQS.**

#### **a. NTshell**

NTshell is a program that runs a DOS batch file on a Unix machine. Special directives within the DQS submission script declare the target: DOS machines and the location of a temporary Samba file system to use. The output is redirected from the remote NT machine to standard output on the Unix machine. The following operations are performed transparently without any user interaction: copy the batch file to the target machine, run it, capture its output, send the output back to the host.

The NTshell program is a shell interpreter in the Unix sense; it can be called by adding “#!” line in a submission script file or directly from the command line. See the documentation in annex C for details.

#### **b. NTbridge**

The NTbridge program links DQS to NT. It simply makes sure that the host and queue files, as well as the core set of environment variables from DQS, are properly exported before running the batch file with NTshell. See the documentation in annex D for details.

In Figure 26, the first 16 queues (named ntnodexx) are the queues linked by the NTbridge to the NT cluster nodes.

## Conclusions and work in progress

We have designed and implemented a prototype of the Experiment Design Submodule the Run Submodule and the Queuing submodule. We have linked these submodules into the Experiment Design and Control Module of our Automated Benchmarking Toolset. These submodules will soon be fully integrated together by a main user interface. We have integrated the DQS scheduling system into our Queuing System Submodule to handle experiment execution on the desired computing platforms, our Linux and NT clusters. To avoid the effort of porting DQS to the NT operating system, we developed a pair of simple tools, called NTshell and NTbridge, to extend DQS to NT.

Based on our initial unit testing, we have devised a number of modifications to both the internal design (add diverse features) and to the user interface (e.g., provide a Command File editor in the Experiment Specification Package, make the GUI of the Experiment Plan Design Package more readable and readable, etc). The completion of this prototype will incorporate these modifications as well as a comprehensive user interface for the overall toolset.

## Acknowledgement

First, I'd like to thank Monique Becker and Alan Mink for giving me the opportunity to get this unique work experience at NIST.

Then the team of the Automated Benchmarking toolset:

Stephane Simon who worked on the performance data storage platform, Guillaume Marcais (Gus) who is still working on the project and who has already worked on the collection, visualization and runtime part of the project. Finally the project supervisor, Michel Courson who has welcome me in the team and has always trusted me during the past height months. His friendly, interactive and efficient way of driving this project has been essential for me. GO TEAM !

Of course, I can't forget my second officemate: Guillaume Lathoud (Java Bert).

I'd like to thank our secretary, Annette Shives who has been so nice with me since the beginning (good night guys!), and Barry Hershman who helped me to maintain the clusters in good condition.

Alan Mink, one more time, has been crucial for the writing of this report (until the last day!).

To conclude, I want to thank my family and my friends for their constant support.



## REFERENCES

- [1] R. P. McCormack, J. E. Koontz, J. Devaney, Seamless Computing with Websubmit, Information Technology Laboratory, National Institute of Standards and Technology, December 23 1998. <http://www.itl.nist.gov/div895/sasg/websubmit/websubmit.html>
- [2] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapam, and T. Newhall. The Paradyn parallel performance measurement tool. IEEE Computer, 28(11):37-46, November 1995.
- [3] D. Reed et al. Scalable performance analysis: The Pablo performance analysis environment I. C. Press, editor, Proc. Scalable Parallel Libraries Conference, pages 135-142, Los Alamitos CA, 1993
- [4] Tcl8.0/Tk8.0 Manual, <http://www.sco.com/Technology/tcl/man-8.0/contents.html>.
- [5] S. Simon, M. Courson, The Cluster Profiling Project, Information Technology Laboratory, National Institute of Standards and Technology, 1998. <http://www.cluster.nist.gov>.
- [6] G. Marcais, A. Mink, M. Courson, An automated benchmarking tool : the analysis and visualization module, Information Technology Laboratory, National Institute of Standards and Technology, 16 August 1999. <http://www.cluster.nist.gov>.
- [7] DQS, The Distributed Queuing System, <http://www.scri.fsu.edu/~pasko/dqs.html>  
D. Duke, T. Green, J. Pasko, Research Toward a Heterogenous Networked Computing Cluster: The Distributed Queueing System, Version 3.0, Supercomputer Computations Research Institute, Florida State University, March 1994.
- [8] NAS Benchmark, <http://www.nas.nasa.gov/Pubs/TechReports/RNRreports/dbailey/RNR-94-007/html/npbspec.html>, May 1994.
- [9] Don Libes, Exploring Expect, O'REILLY, November 1996.
- [10] NIST : <http://www.nist.gov>.
- [11] Joseph A. Kaplan, Michael L. Nelson, A Comparison of Queueing ,Cluster and Distributed Computing Systems, NASA Langley Research Center, June 1994. <http://techreports.larc.nasa.gov/ltrs/>.
- [12] DQS Batch Queueing system, <http://www.th.physik.uni.frankfurt.de/DQS>.
- [13] Professor R Hynds, Serial Batch Work in a Workstation Cluster Environment, [http://www.jtap.ac.uk/uti/projets/h/serial\\_b.htm](http://www.jtap.ac.uk/uti/projets/h/serial_b.htm).
- [14] Mark Baker, Computing Framework Survey: Results, [http://www.sis.port.ac.uk/~mab/Survey/Survey\\_results.html](http://www.sis.port.ac.uk/~mab/Survey/Survey_results.html), 19 August 1997.
- [15] Système de gestion de file d'attente (Batch), <http://www.cch.loria.fr/documentation/batch/DQS/dqs.html>.
- [16] PBS: Portable Batch System, <http://pbs.mrj.com>.
- [17] George E.P. Box, William G. Hunter, Statistics for Experimenters, an intro to design, data analysis, and modeling building.
- [18] The university of Californai at Davis, UCD-SNMP, <http://ucd-snmp.ucdavis.edu>.
- [19] A Mink, "Operating Principles of Multikron Virtual Counter Performance Instrumentation for MIMO Computers", NISTIR 5743, November 1995.
- [20] <http://www.nist.gov/itl/div895/cmr/mpiprof/index.html>.
- [21] <http://www.itl.nist.gov/div895/sasg/websubmit/websubmit.html>.

## Annex A. Nist Overview

### A Brief History of NIST [10]

The National Institute of Standards and Technology (NIST), formerly the National Bureau of Standards (NBS), was established by Congress in 1901 to support industry, commerce, scientific institutions, and all branches of Government. For nearly 100 years the NIST/NBS laboratories have worked with industry and government to advance measurement science and develop standards.

NBS was created at a time of enormous industrial development in the United States to help support the steel manufacturing, railroads, telephone, and electric power, all industries that were technically sophisticated for their time but lacked adequate standards. In creating NBS, Congress sought to redress a long-standing need to provide standards of measurement for commerce and industry and support the "technology infrastructure" of the 20th Century.

In its first two decades, NBS won international recognition for its outstanding achievements in physical measurements, development of standards, and test methods -- a tradition that has continued ever since. This early work laid the foundation for advances and improvements in many scientific and technical fields of the time, such as standards for lighting and electric power usage; temperature measurement of molten metals; and materials corrosion studies, testing, and metallurgy.

Both World Wars found NBS deeply involved in mobilizing science to solve pressing weapons and war materials problems. After WWII, basic programs in nuclear and atomic physics, electronics, mathematics, computer research, and polymers as well as instrumentation, standards, and measurement research were instituted.

In the 1950s and 1960s, NBS research helped usher in the computer age and was employed in the space race after the stunning launch of Sputnik. The Bureau's technical expertise led to assignments in the social concerns of the Sixties: the environment, health and safety, among others. By the Seventies, energy conservation and fire research had also taken their place at NBS. The mid-to-late 1970s and 1980s found NBS returning with renewed vigor to its original mission focus in support of industry. In particular, increased emphasis was placed on addressing measurement problems in the emerging technologies. Many believe that the Stevenson-Wydler Act implemented, throughout the federal laboratories, the practices that had been developed at NBS over the years: cooperative research and technology transfer activities.

The Omnibus Trade and Competitiveness Act of 1988 -- in conjunction with 1987 legislation -- augmented the Institute's uniquely orchestrated customer-driven, laboratory-based research program aimed at enhancing the competitiveness of American industry by creating new program elements designed to help industry speed the commercialization of new technology. To reflect the agency's broader mission, the name was changed to the

National Institute of Standards and Technology (NIST). These efforts, and the organizational changes brought by the NIST Authorization Act for 1989 which created the Department of Commerce's Technology Administration to which NIST was transferred, served as a critical examination of the role of NIST in economic growth. These mission and organizational changes, initiated under the Bush Administration were reaffirmed and strengthened by the Clinton Administration.

In addition to the reviews by Congress, the Administration, and the Department of Commerce, the Visiting Committee on Advanced Technology (VCAT) of NIST reviews and makes recommendations regarding the general policy, organization, budget, and programs of NIST. The VCAT holds four business meetings each year with NIST management, and summarizes its findings each year in an annual report that is submitted to the Secretary of Commerce and transmitted by the Secretary to Congress.

NIST's four major programs are designed to help U.S. companies achieve their own success, each one providing appropriate assistance or incentives to overcoming obstacles that can undermine industrial competitiveness. The programs are:

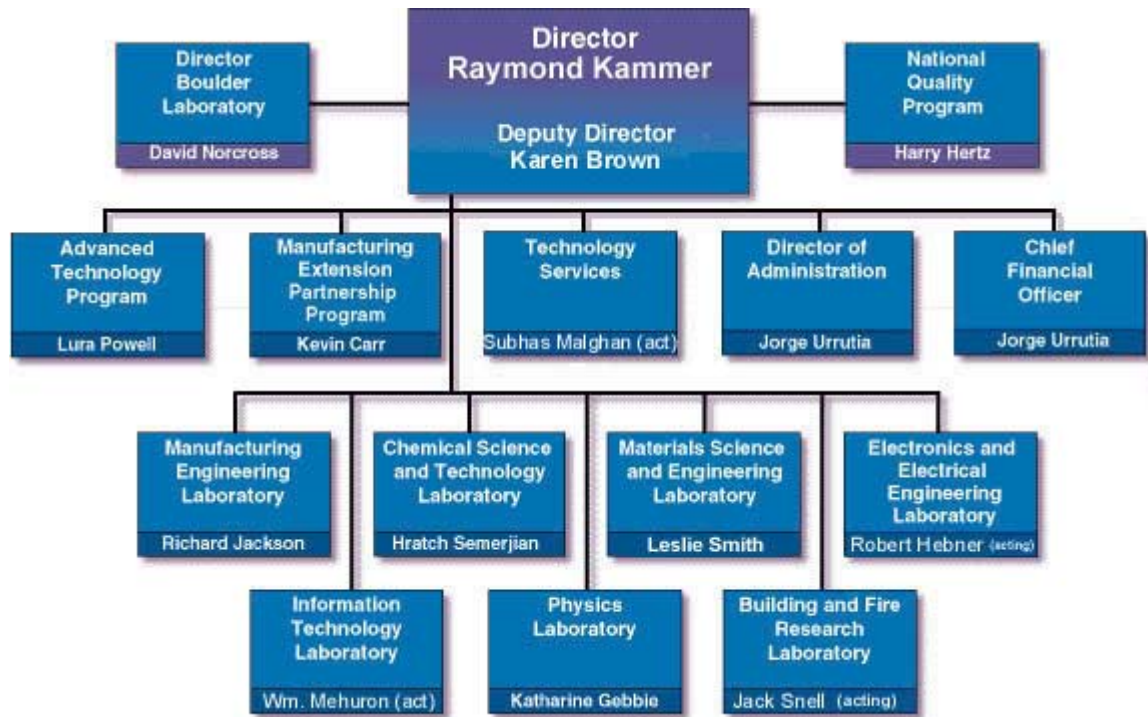
- Measurement and Standards Laboratories that provide technical leadership for vital components of the nation's technology infrastructure needed by U.S. industry to continually improve its products and services;

- a rigorously competitive Advanced Technology Program providing cost-shared awards to industry for development of high-risk, enabling technologies with broad economic potential;

- a grassroots Manufacturing Extension Partnership with a nationwide network of local centers offering technical and business assistance to smaller manufacturers; and

a highly visible quality outreach program associated with the Malcolm Baldrige National Quality Award that recognizes continuous improvements in quality management by U.S. manufacturers and service companies.

NIST organization chart



**Figure 28: Organization chart.**

The Information Technology Laboratory divides itself in division and then in groups. I was working in the Scalable Parallel Systems & Applications in the high-performance Systems & Services Division.

#### Information Technology Laboratory

An agency of the U.S. Department of Commerce's Technology Administration, the National Institute of Standards and Technology's primary mission is to promote U.S. economic growth by working with industry to develop and apply technology, measurements, and standards. The NIST's Information Technology Laboratory (ITL) is responding to the growing need for measurement and testing technology to support the development of computing and communications systems that are usable, scalable, interoperable, and secure. This need has come into sharper focus in recent years with the national effort to develop an information infrastructure and to support U.S. industry in a global information marketplace.

ITL has programs in three major areas:

- developing tests for human-machine interfaces, software diagnostics and performance, computer and network security, advanced network technologies, mathematical software, and conformance to standards;
- collaborating, consulting and operational services in computational sciences and information services; and 3.federal computer and network security activities;
- federal computer and network security activities.

#### High-Performance Systems & Services Division

The High Performance Systems and Services Division (895) of the Information Technology Laboratory enables effective application of high performance computing and communications systems in support of the U.S. information technology industry and NIST by: Conducting research, development and evaluation of advanced hardware and software components, new architectures, novel application technologies, and innovative measurement and test methods for improved computing performance, scalability, functionality, interoperability, flexibility, reliability and economy;

Serving as a testbed for R&D in high-performance computing and information technologies such as embedded computing, displays, and data storage, gaining experience in the deployment of these technologies, and developing metrics for the representative technologies;

Serving as a responsive, effective mission-critical resource spanning computational, communication, mass storage, security, archival, and scientific visualization services; and Providing and managing state-of-the-art computing and networking facilities which integrate and support an enterprise-wide heterogeneous information technology environment for NIST.

## **Annex B. Example of code**

The following 20 pages (condensed) of code listings represents about 30% of the total code I produced for this project.



## Annex C. NTshell

NTshell is a simple Expect [9] script that takes a batch file and runs it transparently on an NT (or any DOS box with a telnet and SMB server) machine, reproducing the output of the script ran on the target box directly to standard output. It can be used as a shell in the `#!` line of a script.

### Requirements:

In order to work, NTshell requires:

- a running telnet server on the target (tested only with TelnetD from Pragma Systems Inc.)
- an SMB share directly accessible from the target machine (for file transfers)
- smbclient (from the Samba package).

Note: the login information (username, password and domain) is assumed identical for both resources.

### Configuration:

NTshell reads its configuration information (target file, share...) directly from the script with specially formatted comment lines or from the command line.

In the batch file, NTshell looks for lines matching this syntax:

```
#N <varName> <value>
```

The command line syntax is:

```
NTshell [-d varName=value] [-D varName=value] [-v] -- file1.bat  
file2.bat ...
```

Note that a `#N` directive or `-d` appends to the current value if it exists, whereas `-D` overrides any previous declaration. Command line declarations precede any declaration in the batch file.

Minimum configuration:

host : name of the remote NT host

userName, password, domainName: login information for remote host and share.

sharePath: UNC path to SMB share to be used, eg. `\\seth\MPIPro`

Additional variables:

exportEnv: comma- or space-separated list of environment variables to be exported to the remote machine at runtime.

exportFile: comma- or space-separated LOCAL,REMOTE pair. Local should be the absolute path to the local file to export; remote must be a filename (no path!!) for the remote end. On the remote side, an environment variable is

declared with the same name containing the absolute path of the copy.

timeout: declare a timeout value in seconds. By default, there is no maximum time. This may be useful for programs that may hang (like MPIPro's mpirun if a job is already running on the nodes).

### Implementation details:

- Comments may either use the DOS "REM" format or be the usual Unix "#" line; NTshell converts them all into DOS REMs before transfer.

- The temporary files are copied to and from `sharePath\tmp\PID[.JOB_ID]\`

- Because "cd" and other NT commands do not support UNC paths, NTshell always maps \$sharePath to the Q: drive. Consider this letter RESERVED.

- NTshell tries to gracefully kill the job upon receipt of a SIGINT or SIGUSR2 by sending CTRL+C and CTRL+BREAK to the command prompt. Try either signal before using SIGKILL, or else the job will keep running on the NT side. In particular, use the '\$\$ -notify' directive in your DQS job so that dqs\_execd send a SIGUSR2 before killing the current job.

## Annex D. NTbridge

This program makes the link between DQS and NTshell. Although a simple declaration in the “#!” line of batch files will do<sup>1</sup>, for the user’s convenience it should be declared as the shell for the virtual queues. It requires a modified version of DQS that generates a queue file (QUEUES\_FILE), similar to the host file but containing the list of (logical) queues in the pool. A patch to DQS 3.2.7 is available in the distribution.

NTbridge simply makes sure that the host and queue files as well as the core set of environment variables from DQS are properly exported before running the batch file with NTshell. Note that the path to NTshell is hard-coded to /usr/local/bin/NTshell but the NTSHELLPATH environment variable overrides this value.

NTbridge calls NTshell with this syntax:

```
NTshell -D host=<host> -d exportFile=$env(HOSTS_FILE),HOSTS_FILE \
      -d exportFile=$env(QUEUES_FILE),QUEUES_FILE \
      -d
exportEnv=QUEUES_FILE,NUM_HOSTS,DQS_CELL,LOGNAME,JOB_NAME,QUEUE,JOB_ID\
[-v] -- <args>
```

NOTE: the NT batch file must use %QUEUES\_FILE% (logical queues) for host file, rather than %HOSTS\_FILE% (physical machines that mirror the NT nodes, most likely all the same).

Example:

```
#!/bin/sh /usr/local/bin/NTshell
# This is a comment line

# These are NTshell directives
#N host ntnode4
#N userName Mpiuser
#N password Mpiuser
#N domainName DIV895-01
#N sharePath \\seth\MPIPro
#N timeout 1000

# These are DQS directives.
#$ -j y
#$ -l qty.eq.2

echo Running from:
hostname

echo Number of hosts: %NUM_HOSTS%
type %QUEUES_FILE%
echo Running MPI Program
mpirun -np 2 -mach_file %QUEUES_FILE% \\seth\MPIPro\ft.A.2.try
COPY c:\mpiprofile.out
\\seth\MPIPro\results\mpiprofile.%JOB_ID%.out
```

---

<sup>1</sup> The current version of DQS (3.2.7) does not handle “#!” lines properly: scripts are always run with the shell declared in the queue configuration.



KNOWN BUGS:

- Need to do a DOS->Unix text conversion (0x0D,0x0A maps to empty lines).
- Killing is barely more than a kludge and does not always work.